# Implementing Responsive Lightweight In-Page Editing

Michael J Rees

School of Information Technology
Bond University
Qld 4229 Australia

Email: mrees@bond.edu.au

## Abstract

The earliest web browsers allowed users to edit and save a new version of any page they were reading. At the start then, the Web provided fully collaborative pages. Mosaic, the browser that popularised the Web in 1993, allowed pages to be viewed but not changed. Specialised shared pages and servers needed to be developed before fully collaborative web pages were available once more. These web page collaboration tools are now widely available but suffer from a variety of drawbacks described more fully in the body of this paper.

The lightweight in-page editing implemented as the Sparrow Project at Xerox PARC offers an excellent compromise solution to collaborative web pages. At the same time the technique can be applied to many types of page content, making it widely applicable. Unfortunately, Sparrow suffers from long round-trip server delays when operating the user interface. The work described in this paper takes some of the basic ideas for collaborative pages from Sparrow but implements them in a highly usable, responsive fashion. This is largely achieved by exploiting the DHTML document object model and client-side scripting. Examples of this new Pardalote system are presented to show the benefits of lightweight in-page editing. Some improvements to the server side page management are also described. Finally, future extensions to Pardalote are discussed, which can take lightweight in-page editing in new and more powerful directions.

Keywords: lightweight editing, in-page editing, user interface design, computer supported cooperative work, HTML behaviors, implementation architecture

## Introduction

Tim Berners-Lee designed and implemented the first web browser on a Next machine when the World Wide Web (Web) was born at CERN in 1990. With this web browser a user could edit and save a new version of any page being displayed. Usually, a new page was created that had to be linked from other pages. Nevertheless, early Web pages were fully collaborative.

In 1993, the ground-breaking Mosaic browser from NCSA was responsible for the Web

becoming the universal data repository it has become today. Unfortunately, Mosaic is a read-only browser, but has become the model for almost every commercial browser in use today. The original collaborative nature of web page browsing was lost. To be fair, the first versions of Mosaic did include facilities to add personal annotations to each page (stored on the user's own machine) and group annotations (annotations stored on a server and available to a group of users). Even these annotation facilities disappeared from the modern commercial browsers.

Since the mid-1990s, several key examples of web page-based collaborative systems have been developed. Even today, new examples of such systems continue to appear. Many are extensive systems with extensive feature lists for shared web page document management and virtual meeting support. A survey of some of these systems is to be found in [HREF1]. Many trace their origins to the seminal meeting of the Collaboration, Knowledge Representation and Automatability working group [HREF2] of the World Wide Web Consortium (W3C).

Most existing web-based systems that result in collaborative pages fall into one or three main categories:

1. Discussion groups. These are typical bulletin board systems that allow users to submit original textual articles to a list. Other users may do likewise, or reply to an existing article. Lists are organised by topic or around specific web pages. A pioneering example is HyperNews [HREF3].
2. Document management based around meetings at a distance. Users at remote locations share and amend meeting agendas, minutes, outcomes and supporting documents. A prime example of this type of system is BCSW [HREF4].
3. Web page editors. Users are presented with pages of interleaved fixed content and text that they can enter and amend. This is a common facility on web sites that host web pages for individual users who are not familiar with more sophisticated web page publishing tools. See Tripod.com for a typical example, but virtually every major web hosting site will have such as tool. A page fragment from Tripod's One-minute Page web page is shown in Figure 1.

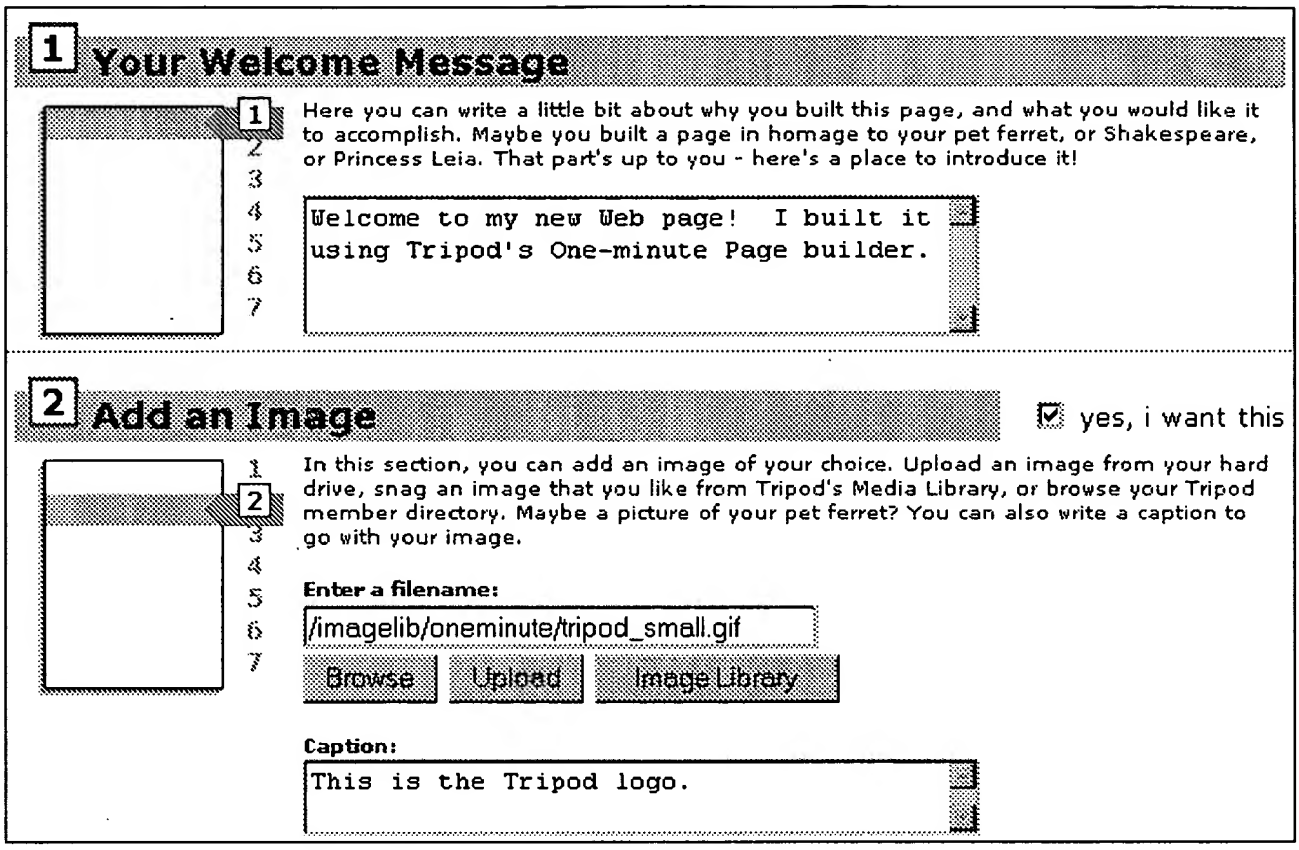


**Figure 1. Part of web page content customisation at Tripod.com.**

All of these systems share the common feature that users are presented with form fields within web pages as can be seen in Figure 1. The text that is entered finds its way into the same page or related pages. The main editing methods employed are:

1. CGI round-trip. Form field contents are sent to a CGI script/servelet/server add-on/ASP page and a new page is generated to be displayed by the user's browser.
2. Java applet. Client side Java classes process the form field contents and communicate in a standard or proprietary manner with either the web server or dedicated server, before returning a new page to the user's browser.
3. ActiveX control/browser add-in: Similar to method 2 but utilising execution elements specific to the browser in use. This method can use asynchronous interaction for the file management access to the server, and thus make the user interface more responsive.

Method 1 is common but typically suffers from long response times, making the user interaction tedious. Java applets used in method 2 overcome the response time problems but suffer from special web server or other server installations and other non-standard server configurations. Method 3 results in a usable user interface but requires the download and installation of special execution elements and maybe server add-ons as well.

Although common, these are not the only methods that can be employed. The PageSeeder application [HREF5] allows discussion items to be submitted by email. The Java web server extracts the text from the email message and inserts a discussion item into the web page. Later versions of PageSeeder have added the more usual fill-in form submission method employing a CGI round-trip.

The importance of shared web pages is best summarised in the concept of the 'empty web'. This concept was aired at length during the Web Education panel session at the inaugural Asia-Pacific Web Conference in September 1998. There, J Dale Burnett put forward the case for the empty web [HREF6]. By this he meant a series of blank web pages in which users can store information of their own choosing, without the need to use sophisticated web publishing tools. In the education context, this gives the user their personal web notebook. The simple act of browsing would allow in situ editing to be shared between student and teacher, and/or other students in the class. Lightweight editing is ideal for this proposal.

## Sparrow lightweight in-place editing

The concept of lightweight in-place editing first appeared as the Sparrow Project described by Bay-Wei Chang in [HREF7]. Chang indicates that Sparrow in-place editing facilitates a different genre of web page: the community-shared page. In essence, the original page author sets the framework for the web page, but designates areas on the page that can be edited by the whole community, which share access to that page. Other parts of the page are static so that the original document structure is retained. In the Sparrow prototype, the community can be as wide as every reader who can display that web page. For a publicly accessible web page, this means all the users on the Internet.

Chang's lightweight editing allows contributors to edit within the browser; edit in-place in the document; edit narrowly, just one item at a time; edit in a structured fashion; edit at a high level of abstraction; and edit collaboratively. This seems restrictive but in practice turns out to be very powerful because:

- All editing is done within the browser; no additional software tools, Java applets, ActiveX controls or add-ins are needed.
- When amending collaborative documents, each user rarely makes major textual changes; rather, minor additions, annotations and incremental contributions are made.

The Sparrow work shows a number of compelling examples of this lightweight editing including to-do lists, bookmarks, email addresses, and general list pages of many types. The applicability of the idea appears almost boundless.

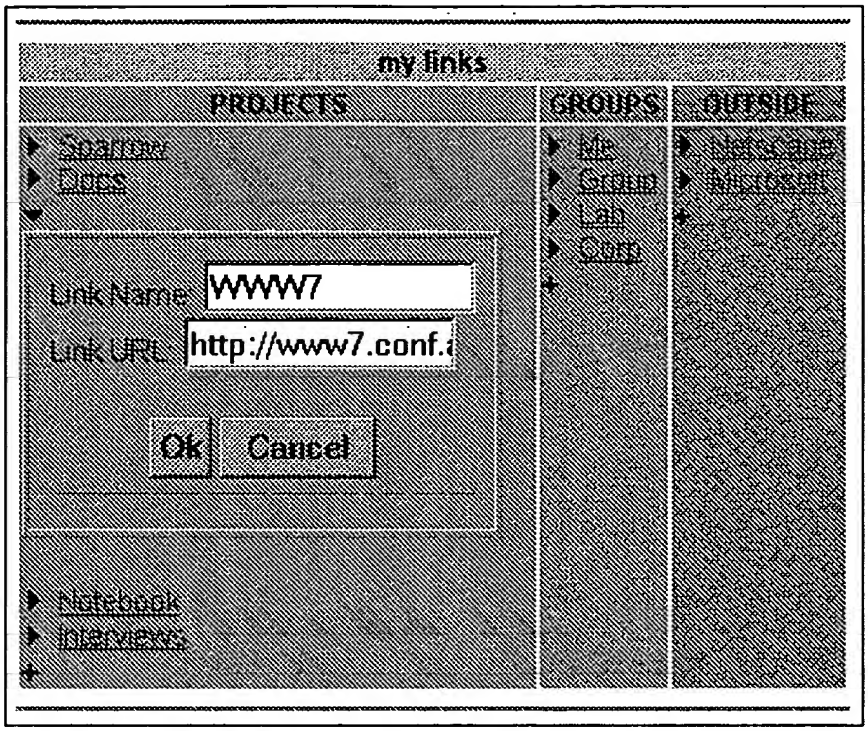


**Figure 2. Part of a Sparrow hotlist page.**

Figure 2 shows an example page fragment taken from Chang's original paper [HREF7]. It shows some powerful features of lightweight editing. First, the lightweight editing symbols are black triangles (pointing right and pointing down) and the bold plus sign. When a user clicks on the right pointing triangle

1. an edit form appears with appropriate editable text fields, and
2. the triangle changes to point downwards.

Now the user may change the contents of the text boxes and click OK to save the changes, which are reflected in the shared web page after a round-trip to the server. Note that the lists of links may be extended by clicking on the plus signs, allowing the user to add a new link to the list. Sparrow has been in use at Xerox PARC from before the publication of [HREF7] in 1998. According to the Sparrow Project web page [HREF8], and to the author's knowledge, there has been no public release of this excellent editing system. For a year or two, the public could access demonstration pages at the Xerox PARC web site to try out the features of Sparrow. At the time of writing, this demonstration editing has been switched off, but the public lightweight editing contributions are still to be seen.

To implement its feature set, Sparrow uses special HTML tags and HTML comments in a particular format to represent the parts of the web page that can be edited. Page authors must insert these Sparrow tags into their pages. This ensures these extra tags are ignored by browsers. Such inclusions can be difficult using commercial web page publishing tools, except by editing the HTML source directly.

To allow wide deployment, the Sparrow tags are interpreted by a series of CGI scripts, and generate the necessary visual cues, forms and buttons. The form submit buttons trigger the script execution and switch between display and editing mode. Using CGI scripts allows

browser independence at the cost of an unresponsive user interface over slow Internet links.

## Requirements of Pardalote lightweight editing

The author of this paper is impressed with the simple, but powerful ideas exhibited by Sparrow. Using the Sparrow demonstrations at Xerox PARC from Australia is less impressive because of lengthy response times. Since Sparrow was developed, a number of new interaction technologies have become available for web applications. These developments allow more responsive user interfaces to be constructed. The author established the Pardalote project to re-implement some of the Sparrow ideas with a more usable interface, and to extend the range of lightweight editing in general.

The requirements of Pardalote lightweight editing are:

| | |
|---|---|
| R1 | Users sharing a web page should do so within a standard browser without the need for s executable modules, yet still have access to a responsive user interface. |
| R2 | Collaborative editing should be confined to the content and position of specified fragme within a page, in a structured fashion determined by the original author of the page. |
| R3 | Creation of collaborative pages should be possible with standard web page publishing tc employing regular features of HTML, style sheets and scripts. |
| R4 | Installation of the style sheets and script files should be convenient for the web site administrators. |

Pardalote requirements R1 and R2 mirror those of Sparrow to a great extent. R3 and R4 aim to increase the usability, portability and wide adoption of Pardalote compared with Sparrow. In addition, R1 includes the need to make a more responsive user interface than Sparrow can provide.

The four requirements recognise the three types of user involved in collaborative web pages, namely:

| | |
|---|---|
| Web site administrators | Users responsible for a creating and maintaining the web site on which Pardalote lightweight editing will operate. |
| Primary page authors | Users who create web pages marked to allow Pardalote lightweight editing. |
| Collaboration group users | Groups of users who collaborate to determine the content of shared web pages. |

It is assumed these three user types possess web page building expertise in decreasing amounts. Most of the systems described above and in [HREF1] put a great deal of emphasis on the collaboration group users. This is exactly as it should be since the user interface for end users is paramount to the success of any software. However, successful deployment of an in-page editing system of any kind relies heavily on the ease of installation in the first place, and the ease of collaborative page creation thereafter. Pardalote aims to treat the first two categories of user with equal reverence.

Note that for ease of reading, collaboration group users are shortened to users or Pardalote users in the remainder of the paper.

## Pardalote lightweight in-page editing mark I

Early experiments with Pardalote implementation put browser independence aside and concentrated on the responsive user interface for collaboration group users, and the convenience of page construction for primary page authors. It was fairly obvious that the dynamic HTML document object model and client-side scripting could allow responsive in-page editing. JavaScript, soon to be ECMAScript, was chosen to make browser independence possible in the longer term.

Pardalote needs just a few lines of JavaScript to enable the in situ editing of simple text entries, radio buttons, checkboxes and selection lists . The original version of Pardalote, shown in Figure 3, demonstrates the in-page editing of a section, a Pardalote page fragment. The only additions to a normal web page are the *Save Changes* button and the in-page editing characters, ✍. Whenever users see this character, they know that editing is enabled.

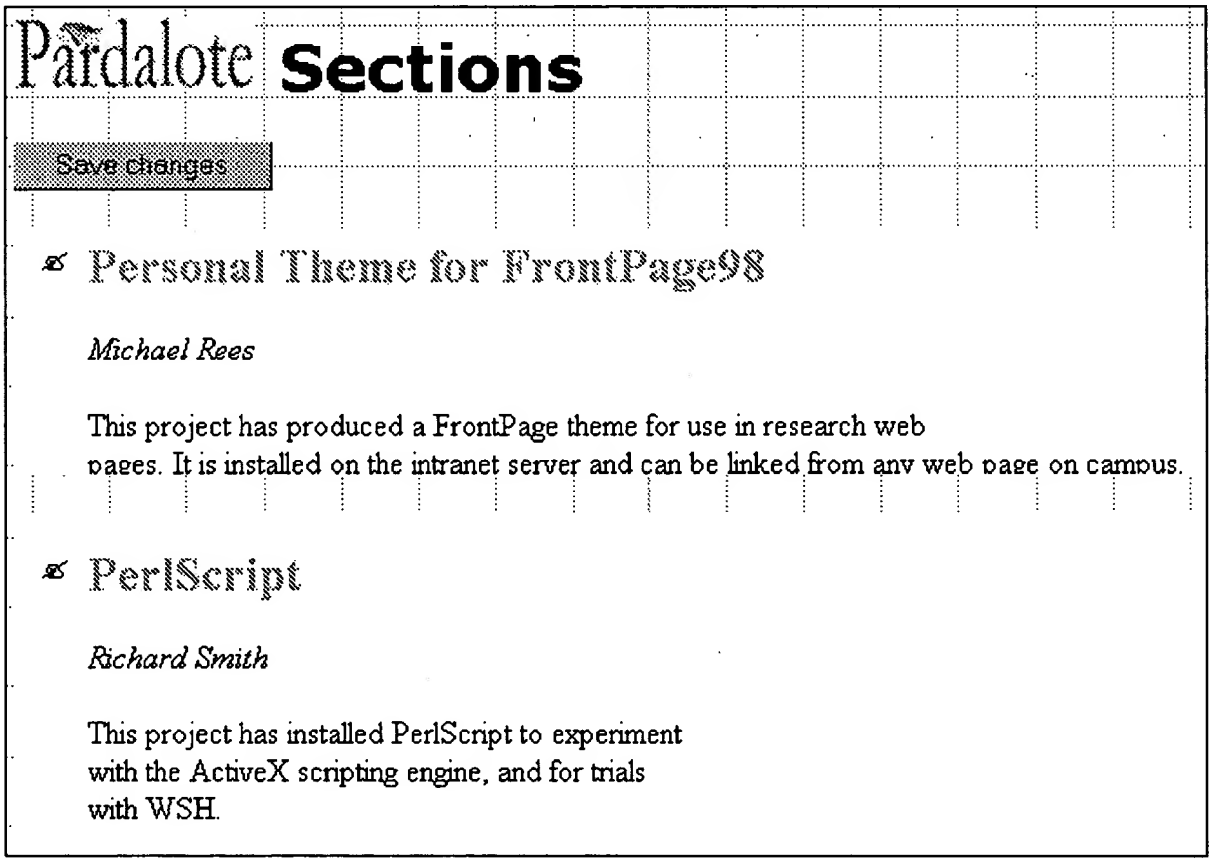


**Figure 3. Pardalote sections, editable page fragments.**

When the user clicks on the heading of a section marked with the in-page editing character, an editing form is revealed as shown in Figure 4. This change occurs immediately, within a fraction of a second, and does not require a round-trip to the server and back as with Sparrow. For space reasons only the section being edited is shown in Figure 4, but note that the remainder of the page is displayed as normal, just the editing text boxes and text areas appear.

Thus the user sees the whole context of the page while editing, and scrolling can occur as normal.

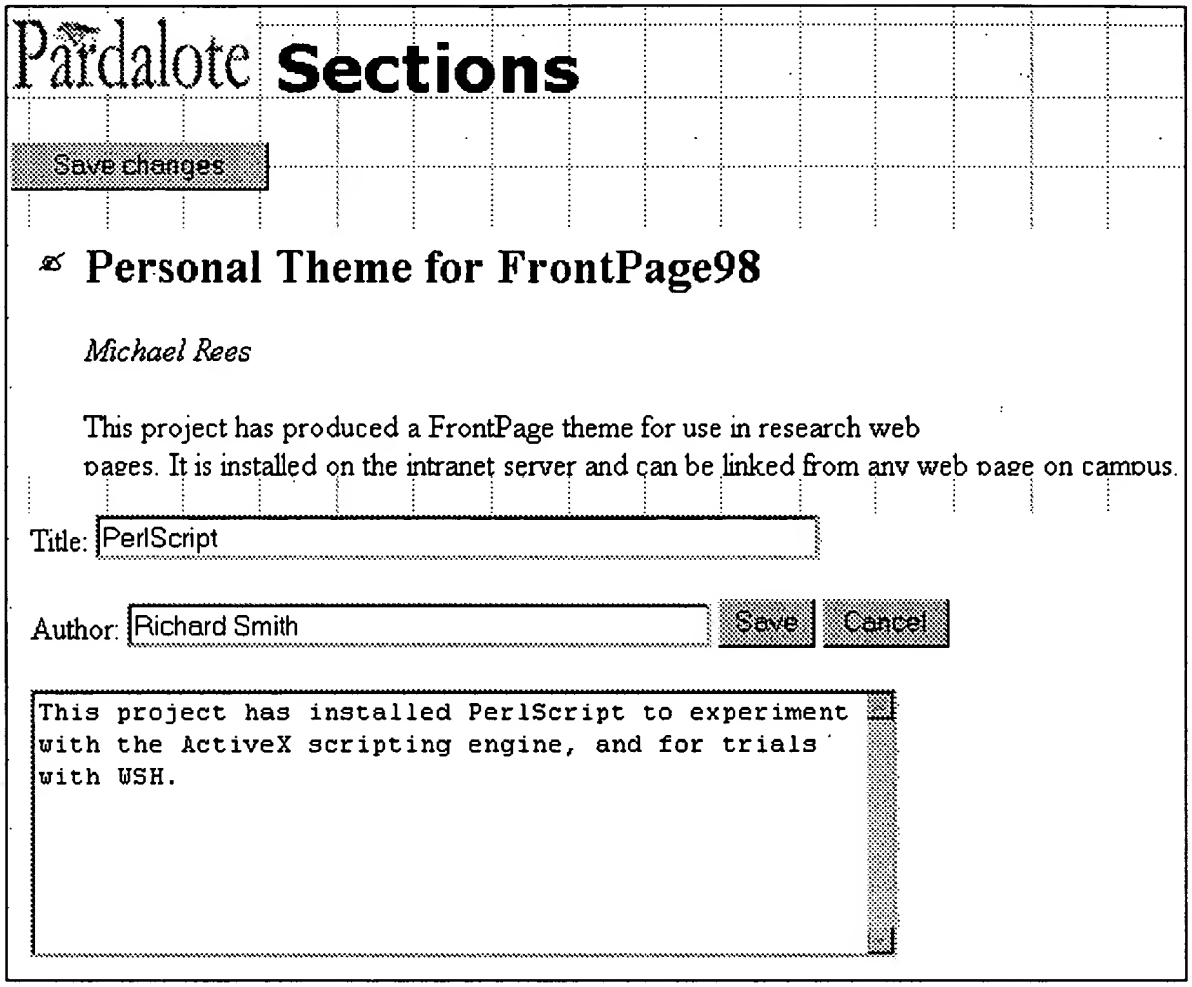


**Figure 4. Pardalote section fragment editing form.**

Clicking on the *Save Changes* button commits the text changes to the page on the web server. In this version of Pardalote and other versions mentioned below, a synchronous, server round-trip is necessary to implement this feature. A possible solution to make this an asynchronous process is discussed later in the paper.

This version of Pardalote employs the Microsoft scriptlet technology. Scriptlets are regular HTML pages containing script tags and HTML tags. The skeleton form of the Pardalote sections scriptlet is:

```
<html>
  ...
<script language="JavaScript">
  ... // Pardalote scripts here
</script>
<body id="section" onload="initScriptlet();">
<div id="read" style="visibility: visible">
  ...
</div>
<div id="edit" style="visibility: hidden">
```

```
  ...
</div>
</html>
```

The sections scriptlet maintains two views of the HTML, one DIV tag for viewing and one DIV tag for editing. Pardalote scripts handle the view switching and the transfer of textual content between the two views using script manipulation of the DHTML document object model. This allows each section in the example shown in Figure 4 to act independently. Lengths of the script functions involved is very small so that view switching and text transfer are very fast, resulting in a more than adequate responsive user interface.

Primary page authors simply insert Pardalote scriptlet objects into their pages as follows:

```
<object id="section1" type="text/x-scriptlet" width="100%">
<param NAME="URL" VALUE="sectionScriptlet.htm">
<param NAME= ... >
</object>
```

Authors simply need to take care over the ids of each section scriptlet object tag. When first inserted in this way the Pardalote sections have empty, default text inserted. As users display and edit the page, they fill in each section with useful text.

Such an implementation allowed experiments with user interface design to be carried out. It is necessary to preserve a 'natural' format in the display view so that the text appears to be a logical part of the page. In edit view, suitable editing and button controls are needed to make the editing actions as simple and convenient as possible.

From an implementation point of view, scriptlets have some benefits. The HTML tags used for the editable page fragment are kept together with the scripts that manipulate them. One can then quickly build scriptlets with various tag structures to fit the needs of the collaboration team. Each scriptlet will have a unique set of scripts, but can be reused many times within the same page and in any number of other pages.

However, scriptlets suffer from script variable and DHTML document object model scope problems. Within scriptlets, it is simple to access the HTML tags in the document object model. For some features, such as creating new editable fragments and moving existing ones, it is necessary to access the document object model for the page as whole, ie outside the immediate scope of the scriptlet. This is difficult to achieve and leads to scripting conflicts. A better approach was needed.

## Pardalote lightweight in-page editing current implementation

The current fully operational version of Pardalote uses Microsoft HTML behavior technology applied to individual regular HTML tags or simple tag sequences. No OBJECT tags are needed. Editable page fragments are indicated by applying regular style classes from a CSS style sheet provided by the Pardalote application. Most web page publishing packages support CSS style sheets, so the primary page author's task is extremely simple and straightforward.

(Again for the sake of brevity, the term 'editable page fragment' is replaced by i-grain, short for information grain. There does not appear to be an agreed terminology for this concept, so i-

grain is introduced to describe this unique form of information. Ideas for further processing of i-grains appear later in the paper.)

To create a Pardalote web page containing i-grains paragraphs, for example, the page author:

1. Links the lweStyles.css style sheet into the page using a link tag in the HEAD section. Part of this style sheet defines the .lwePara class:

```
/* Pardalote Style Sheet
Michael Rees © 1999, 2000
*/
  ...
/* Para Classes */
.lwePara {behavior: url(para.htc)}
  ...
```

2. Insert a one-line script tag in the HEAD section:

```
<script id="lweScriptInclude" src="lweCommon.js"></script>
```

3. Assigns the .lwePara class to all tags that should be paragraph i-grains, for example:

```
<p class="lwePara">This is a paragraph i-grain with full editing features.</p>
```

An ordinary web page, without any Pardalote features, is shown in Figure 5. This page can be created with a wide variety of web page publishing tools.

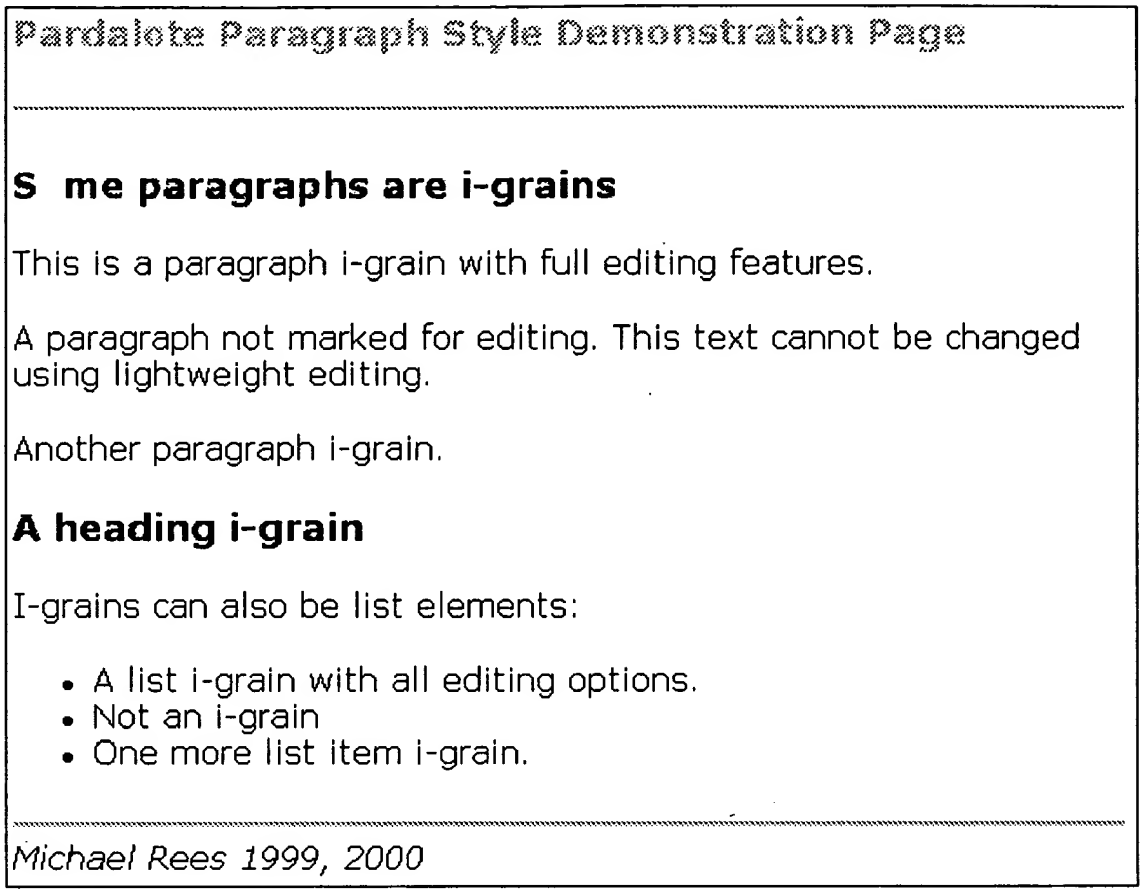


**Figure 5. Regular web page.**

When the page authors performs the steps described above, and loads the same page into the browser, the output of Figure 6 is achieved.

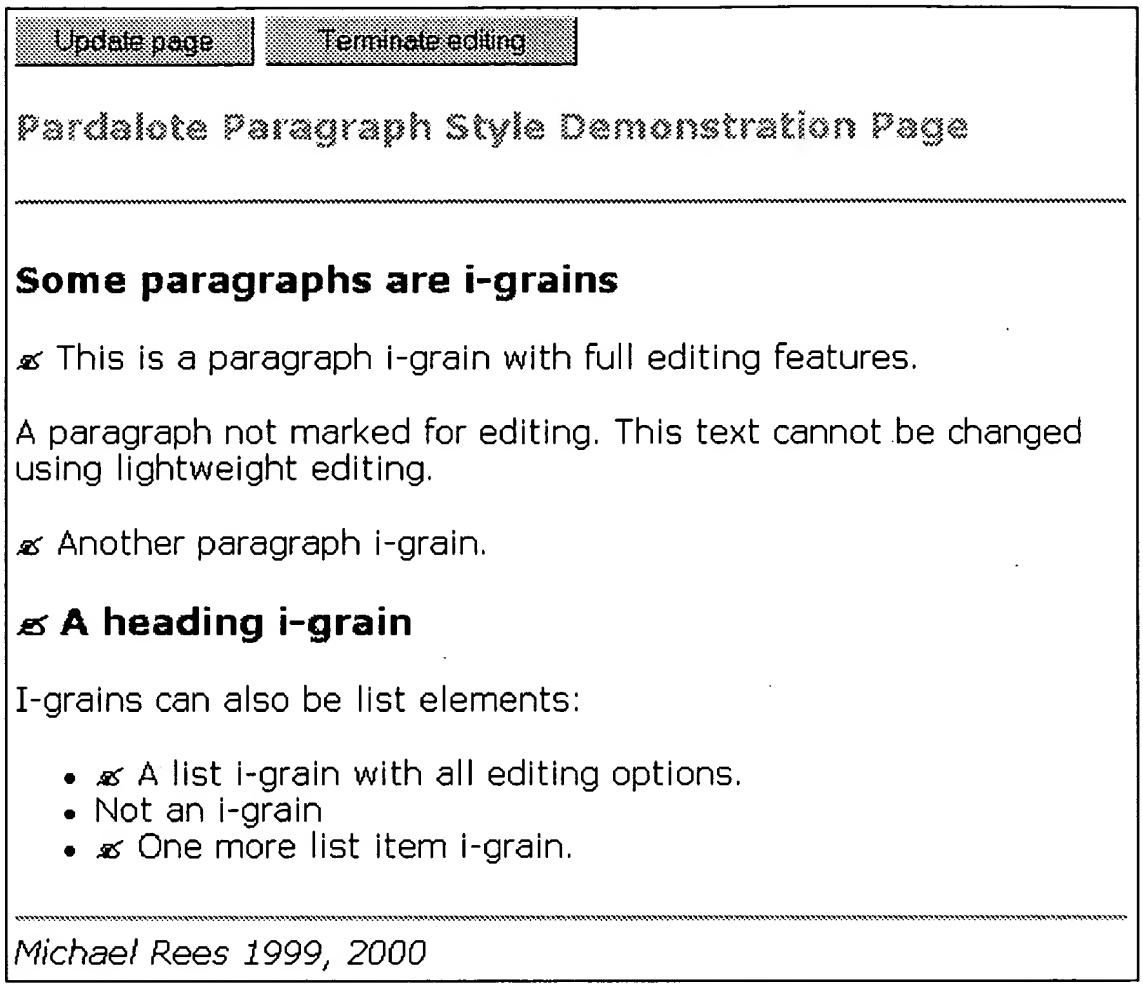


**Figure 6. Web page with Pardalote features added.**

The two differences between Figures 5 and 6 are:

- Two buttons, *Update page* and *Terminate editing*, have been inserted at the top of the page
- Each paragraph i-grain has the Pardalote lightweight editing symbol inserted before the first word

Both of these differences have been added dynamically by Pardalote scripts as the page loads in the browser. The user will also notice a further dynamic change. As the cursor moves over any i-grain text, it will turn into a hand cursor. This indicates the i-grain text is active. Figure 7 shows the instant response that clicking on the first i-grain paragraph elicits.

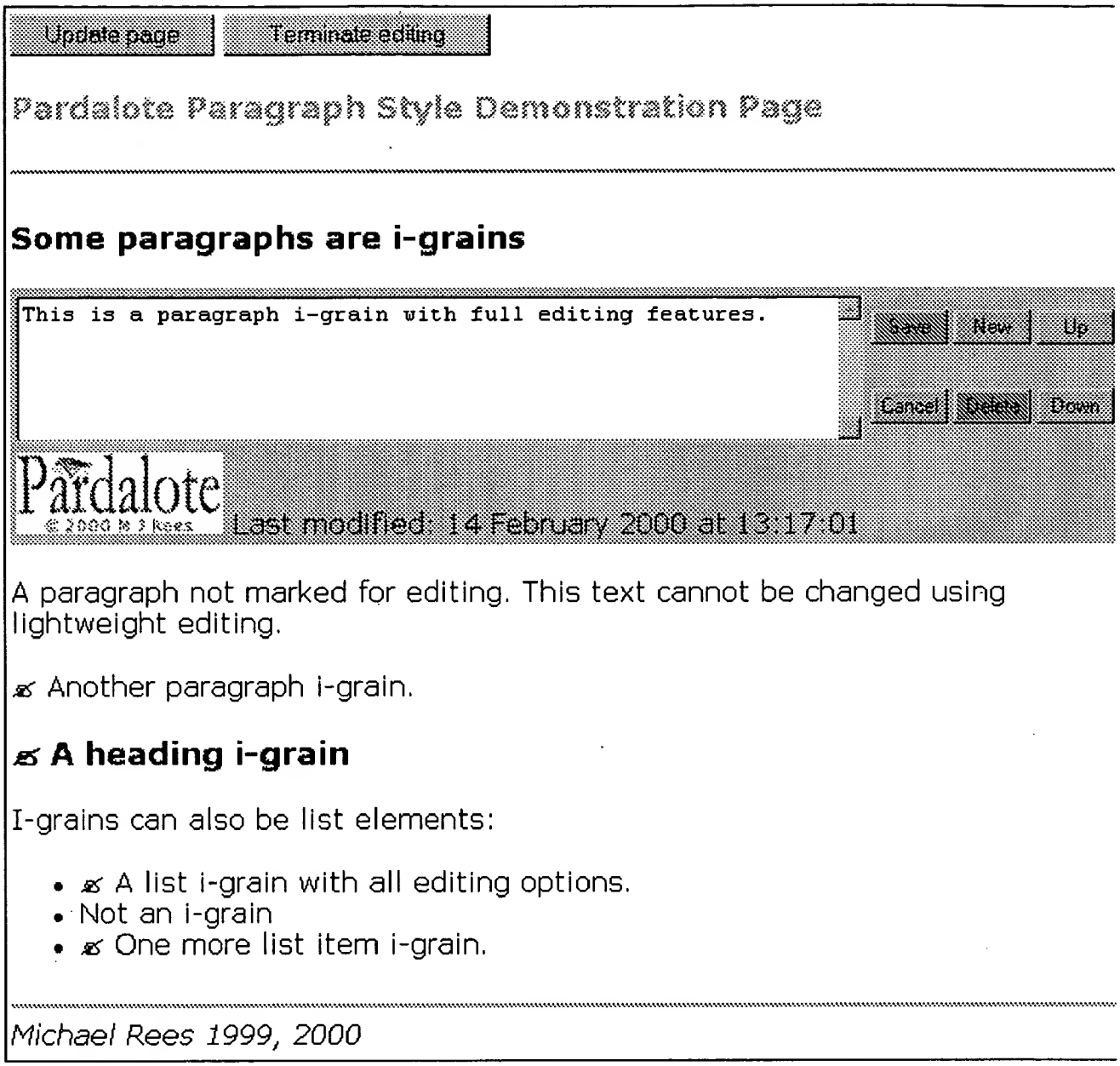


**Figure 7. Activating Pardalote lightweight editing.**

The i-grain is now in edit mode, and provides the user with a range of editing features, as well as cancelling the edit altogether:

- The text of the i-grain can be changed in the text area and saved with the Save button. Changes are reflected in the display view mode similar to Figure 6.
- Clicking the *New* button saves any changes to the current i-grain and inserts a new empty i-grain immediately following the current one. The same tag type is used for the new copy.
- Using the *Delete* button simply removes the i-grain from the page completely.
- The *Up* and *Down* buttons move the i-grain within the page relative to other i-grains with the same tag type. (A number of different movement options are possible here. The exact algorithm used awaits more extensive user testing.)

These simple set of editing features provide for a relatively rich set of page changes. The modifications, though, are still constrained within the basic document structure set by the primary page author. Note that a date stamp is retained with the i-grain to record the date and time of last modification. This data is encapsulated in attributes to the SCAN tag which hold the Pardalote lightweight editing symbol as the first child of the i-grain tag.

Currently, i-grains can either be removed or remain editable for the life of the page collaboration. A freeze option has been added that allows an i-grain to be frozen, ie cease to be an i-grain, but with the text remaining in the page. Freezing as an option is added to an i-grain by using a variant of the normal paragraph i-grain class. Setting the class to lwePara-f causes an extra button to be displayed when this i-grain is edited as shown in Figure 8.

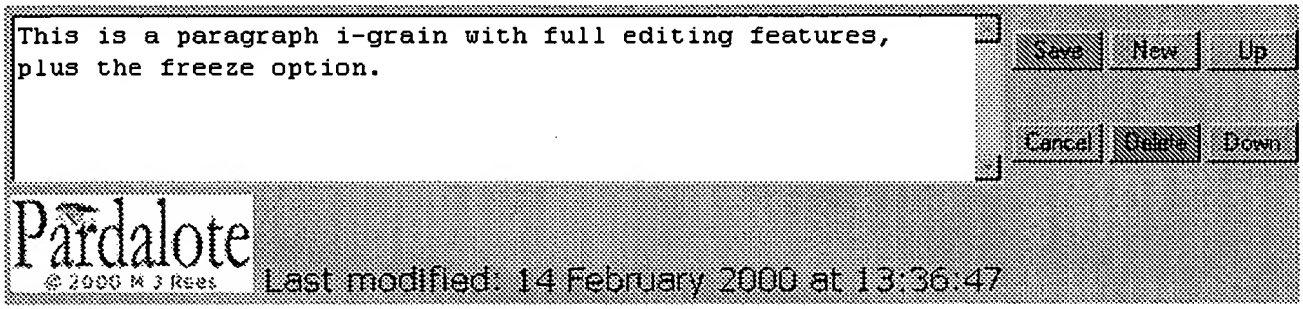


**Figure 8. The i-grain freeze option.**

Returning to Figure 7 and the buttons inserted at the top of a Pardalote lightweight editing page, the *Update page* button sends the current HTML text for the whole page back to an ASP page on the web server. The HTML text contains all the changes made to the i-grains as well as the static contents of the page. Pardalote scripts insert two hidden fields into the form that holds the *Update page* button. One field holds the URL of the page and the other the complete, updated HTML of the page. With the updated HTML text, the ASP page overwrites the original web page file on the server. Thus a server round-trip is required in the current Pardalote application when saving the page.

The *Terminate editing* button performs a global i-grain freeze on the whole web page, removing all Pardalote style sheets, SCRIPT tags and CSS classes from the HTML, before rewriting the web page file. This action returns the web page to its state before Pardalote lightweight editing was added, and the browser display will be similar to Figure 6, except that all intervening i-grain updates will be retained. Normally the primary page author will initiate the page freeze, but in the current version of Pardalote, any user can do this.

The current Pardalote is implemented using the Microsoft IIS web server. All development has been carried out by the author over a few weeks using the Microsoft Visual InterDev web application development tool. InterDev uses the Microsoft FrontPage extensions to manage documents on the web site, and provides a series of editors for HTML, ASP, CSS and script files. With online help files included, only about 15 files are needed to implement Pardalote. The JavaScript in the ASP document that overwrites the web page being edited is all of 8 lines long. Admittedly, no checks are performed for conflicting edits by two of more users. For now, Pardalote is designed for small collaboration teams, where such conflicts are extremely rare. This check must inevitably be incorporated into future versions of Pardalote as discussed in the extensions section below.

## Further i-grain examples

The paragraph i-grain is very versatile, but can sometimes all too much freedom in the ordering of individual paragraphs. There are occasions when sequences of tags need to be kept together and edited as a unit. Following the tradition set by Sparrow and Pardalote mark I, a section i-grain was implemented matching the examples given above. Figure 9 shows part of a page containing three section i-grains, with the second section being edited.

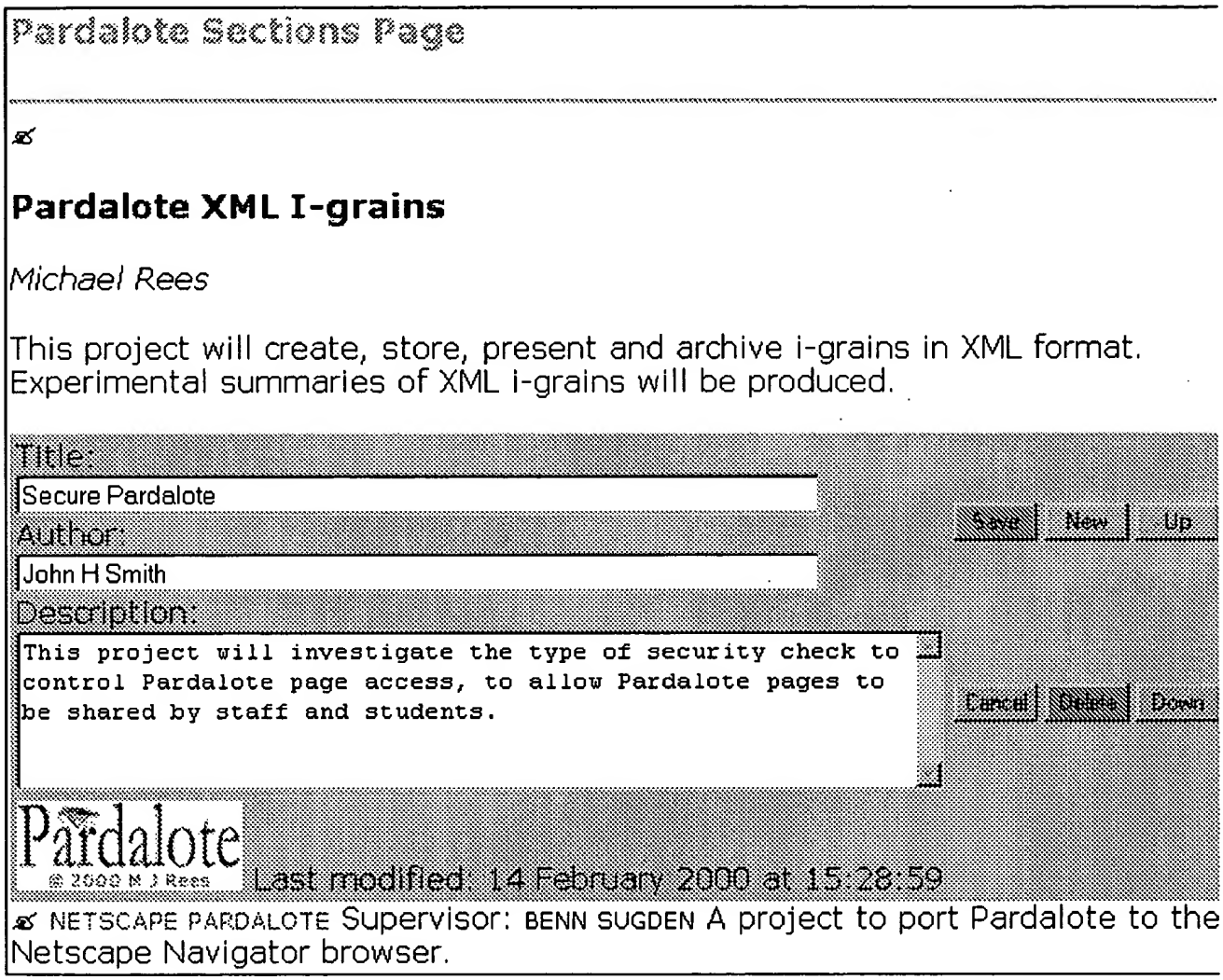


**Figure 9. The section i-grain.**

The page in Figure 9 represents a list of project descriptions, which research staff can edit and extend. Each section i-grain contains three tags capable of holding text, one each for the title, author and description. All section i-grains are contained in a surrounding DIV tag that has class set to lweSection. The i-grain scripts retain the contained tag types and any intervening text, so that a variety of layout structures and text styles can be used. Note the different layouts of the 'Pardalote XML I-grain' project and the 'Netscape Pardalote' projects. All section i-grains will duplicate their own structure and layout when the *New* button is used. A section i-grain with freeze option is available via the lweSection-f class.

Another i-grain type is the link. This allows primary page authors to build pages containing

lists of useful links, and for the collaboration group to aid in their maintenance. Figure 10 shows a fragment of such a page when one of the links, 'Sparrow', is being edited. Users can change the URL and link texts. An extra features of the link i-grain is the ability to actually follow the link. This is necessary because when the page is displayed in Pardalote lightweight editing mode, clicking on the link will activate editing mode rather than follow the link. In the example in Figure 10, the link i-grains are unordered list items within a table cell. The link i-grain style can be attached to any tag that can contain the A tag that actually stores the link. Examples of containers include P, TD, ADDRESS, SCAN and DIV.

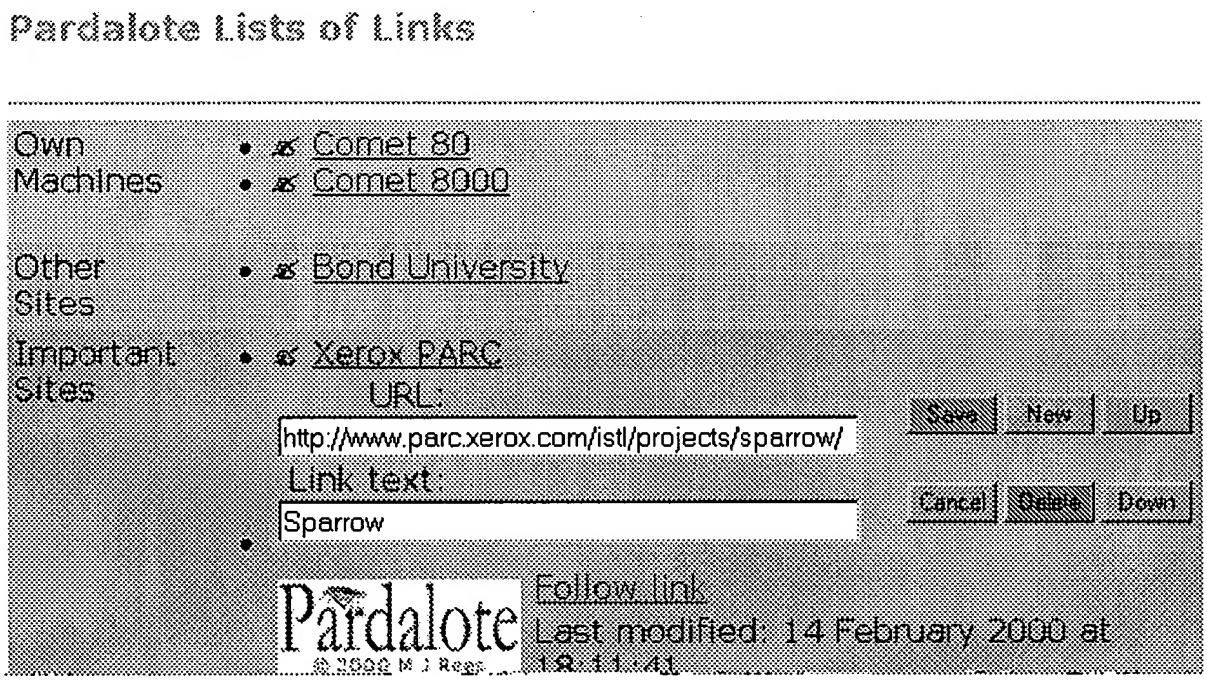


**Figure 10. Examples of the link i-grain.**

As a final example of the versatility of the lightweight editing concept, consider Figure 11, which shows an in/out board. Here the automatic date stamp appears as part of the visible information, and is inserted at the time of saving.

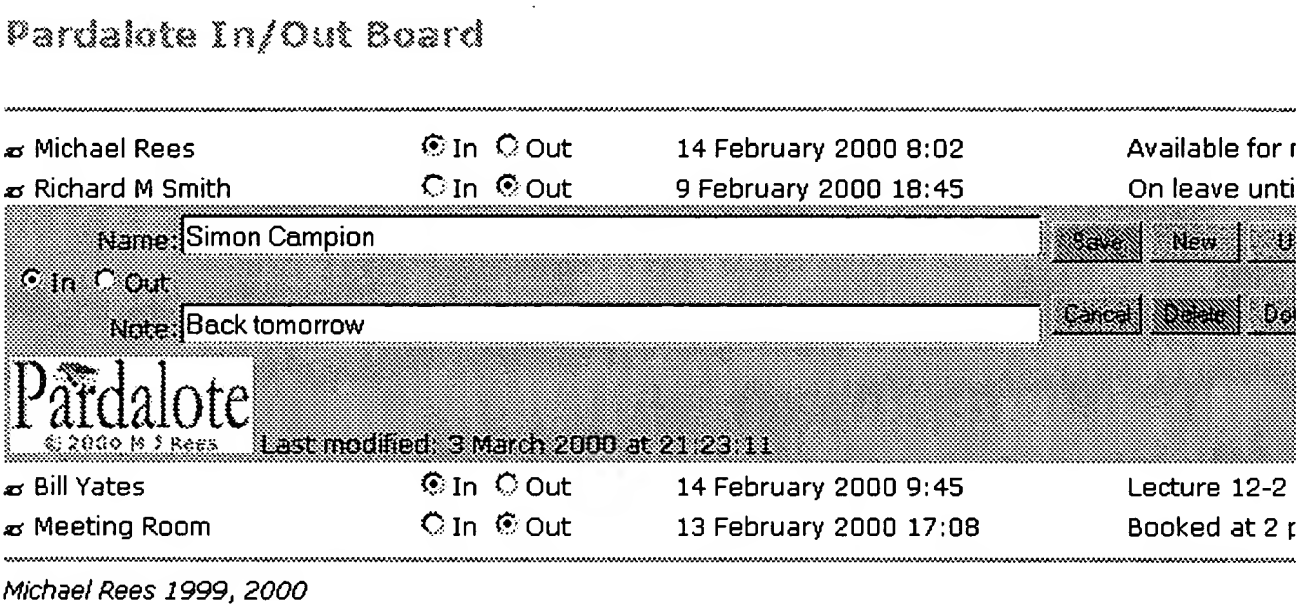


**Figure 11. Example of a composite i-grain, an in/out board.**

At the time of writing, one would have to trust the others on the in/out list and the users who simply need to view this list to collaborate sensibly. Each is able to modify any part of the list, maybe accidentally or maliciously. Obviously, an enforced security system to control access is required.

## Future Pardalote extensions

The simplicity of the i-grain concept will allow it to be extended to encompass several other types of page fragment, including non-textual media. One could imagine exposing a palette of i-grains to be created and move about from any generic i-grain, making the editing process more flexible and powerful. Even whole conference papers might be created from i-grains. Conference program committees might provide a skeleton i-grain paper with authors constrained to insert standard i-grains to enforce the same paper layouts! Future papers in this series will explore this possibility.

Beyond new types of i-grain, the most pressing problem for Pardalote is how to detect and recover from the situation where two or more users attempt to update the same web page at the same time. Sparrow already incorporates such a feature and returns all versions of the i-grain contents to the user to make the choice of the final version. The proposed scheme for Pardalote is somewhat simpler. Where conflict occurs, the user should be warned and another window or windows opened to show the other version or versions. The user can decide to reject any version or attempt to merge the edited versions. Collaboration timescales in practice make sure situations occur very infrequently, as the Sparrow paper points out.

Another extension might be to provide users with a virtually empty page and allow them to construct new layouts. Such an approach is being adopted by the EditLive! product [HREF9]. While relying on ActiveX controls or browser add-ins, EditLive! is nevertheless a powerful in-page editor. To quote from their product description, EditLive! is 'designed for seamless integration with websites and web applications. It can be used for many web editing tasks –

from creating a greeting card on an e-greetings site to editing the local intranet'.

An obvious Pardalote extension would be to allow full DHTML editing of the i-grain contents. Two honours student projects have explored this aspect. One of these projects, Raptor, is described briefly in [HREF1]. Both these projects use the standard DHTML editing component from Microsoft. This approach is used by EditLive! as well. Of course, this immediately ties the use of Pardalote to Microsoft Internet Explorer, but some alternatives exist for the Netscape browser.

At the start of this paper, shared web page discussion groups were discounted as being too restrictive and the useful information too fragmented. However, discussion groups do have some big advantages—they provide an automatic archive of contributed articles and an effective history of contributions. Pardalote and Sparrow produce a final, edited version of a collaboration, in a form that can be distributed to larger groups, or even made public. It would be useful to implement some kind of chronological audit trail or journal which records all contributions, even those that were discarded later. Not only should each i-grain be structured in the journal, but the journal itself must be structured for easy search, presentation and subsetting. An XML representation will be entirely appropriate here. Indeed, one can envisage i-grains being represented as XML data islands in HTML pages, and the whole lightweight editing process operating on the XML form. Future versions of Pardalote will explore this extension. It should be noted in passing that the in-page discussions provided by PageSeeder [HREF5] already employ XML for this purpose.

Finally, Pardalote needs to incorporate levels of secure access to certain pages and editing choices on those pages. One can envisage making a page in the process of being edited, the In/Out board for example, being read-only for the browsing public, and editable only by a select group of users. Web page access is a global problem and a number of secure access solutions are already available. Pardalote can exploit these solutions using an editing agent solution. The Pardalote editing agent is a web page in a secure part of a site, only accessible by the group of users allowed to edit a Pardalote page. The Pardalote page itself is read-only and accessible to public browsing. To edit a Pardalote page, a user accesses the editing agent page after authentication by the chosen security method. The editing agent page creates a temporary page with Pardalote editing enabled. The user makes the necessary changes, and then uses the existing *Terminate editing* facility to write the page back to the public web folder.

## Conclusions

The current version of Pardalote is a fully functional lightweight in-page editing system. Pardalote already provides both fine-grained and course-grained editing control of web pages of many different types. For the features described above, only 12 files totalling less than 30 Kbytes need be installed on each web site to make use of Pardalote editing. The files consist of a style sheet, 6 JavaScript files, 4 DHTML behavior files and one ASP page. A few extra HTML are used to provide online help information for the users. Installation for the web site administrator is almost trivial. These files can be reused across hundreds or thousands of pages. At the time of writing, Pardalote runs only on Microsoft Internet Explorer and IIS server. However, because no client-side executable components are used, it will be perfectly possible to extend the JavaScript files to work with Netscape Navigator.

During the lightweight editing process, Pardalote is easy and responsive to use. Only page

saving is prone to server round-trip delays. A new version of Pardalote is being tested which uses remote scripting to make page saving asynchronous. Then, page saving will involve no delays at the user interface, and users will be able to continue editing while the file is rewritten at the web server.

To date, the Pardalote system has been tested by close research colleagues of the author. Some Pardalote pages for day-to-day use are currently being deployed within the author's organisation for use by small collaborative teams. A small set of survey form pages employing Pardalote are soon to be released for use by a group of external users. The results of these deployments will be reported in further papers. Plans are train to show Pardalote to commercial users as well.

The next research challenges that await are concerned with how best to exploit the Pardalote editing, what new i-grains are needed, and to see how it withstands real users. Since no knowledge of HTML is required, one can expect virtually all users with minimum computer skills to be able to use Pardalote productively. Later papers will report whether this is correct.

(Reviewers: Should this paper be published at AusWeb2K, I would expect not only to demonstrate Pardalote during a presentation, but also have a public web site demonstration available for all interested parties to try.)

## Hypertext References

HREF1
Rees, M J, 'User Interface for Lightweight In-line Editing of Web Pages', Proceedings of the 1st Australasian User Interface Conference AUIC 2000, Canberra, January, 2000, pp 88-94, also at http://comet.it.bond.edu.au/pardalote/auic2000/ReesAUIC2000.htm.

HREF2
Workshop on WWW and Collaboration, MIT, September 11-12, 1995, http://www.w3.org/Collaboration/Workshop/.

HREF3
HyperNews Home Page, http://www.hypernews.org/HyperNews/get/hypernews.html.

HREF4
Basic Support for Cooperative Work (BSCW) Home Page, http://bscw.gmd.de/.

HREF5
PageSeeder, Weborganic Pty Ltd, http://www.weborganic.com/.

HREF6
Asia Pacific Web Conference (APWeb98), Beijing, China, September 27-30, 1998, http://www3.cm.deakin.edu.au/apweb98/.

HREF7
Chang, B-W, 'In-Place Editing of Web Pages: Sparrow Community-Shared Documents', WWW7 Conference, Brisbane, 1998, http://www7.scu.edu.au/programme/fullpapers/1929/com1929.htm.

HREF8
Sparrow Project, Xerox PARC, http://www.parc.xerox.com/istl/projects/sparrow/.

HREF9
EditLive!, Ephox Pty Ltd, http://www.editlive.com/.

## Copyright



---

[ Proceedings ]

---

*AusWeb2K, the Sixth Australian World Wide Web Conference, Rihga Colonial Club Resort, Cairns, 12-17 June 2000 Contact: Norsearch Conference Services +61 2 66 20 3932 (from outside Australia) (02) 6620 3932 (from inside Australia) Fax (02) 6622 1954*